On compilers for FPGA accelerators

Christophe Alias

Inria, LIP/ENS-Lyon, CNRS, UCBL

Workshop Inria/EPFL – January 9-10, 2020

CASH Inria Team

Compilation and Analysis for Software and Hardware
Joint Inria Team @ LIP, ENS-Lyon

Compile energy efficient software and hardware for high-performance computing

Permanent members

- **Christophe Alias (CR Inria):**
  - high-level synthesis, compilation, polyhedral model.
- Laure Gonnord (MCF Lyon 1):
  - abstract interpretation, compilation, semantics.
- Ludovic Henrio (CR CNRS):
  - programming languages, actors, semantics.
- Matthieu Moy (MCF Lyon 1):
  - hardware simulation, many-core, dataflow languages.

High-Performance Computing: Growing Challenges

- Power-efficiency
  $\leadsto$ New kind of accelerators (CPU $\rightarrow$ GPU $\rightarrow$ FPGA)
- Data movement is a major energy bottleneck
  $\leadsto$ Optimize communication and computation
- Programming model: parallel programming is a pain
  $\leadsto$ Extract efficient parallelism

Goal: Optimized compilation for HPC applications
Target: software & hardware

Our "end-users"

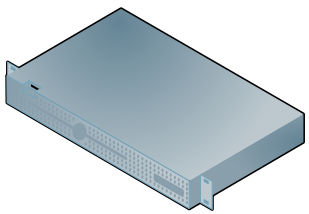
Gas prospector

Application developer

Compute kernel

```
for i := 1 to N − 2
  for j := 1 to N − 2
    Gx := (C[i+2,j+1]+C[i+2,j]+C[i+2,j+2])
      −(C[i,j+1]+C[i,j]+C[i,j+2]);
    Gy := (C[i+1,j+2]+C[i,j+2]+C[i+2,j+2])
      − (C[i+1,j]+C[i,j]+C[i+2,j]);
    B[i, j] := √((2Gx)² + (2Gy)²;
```

CASH

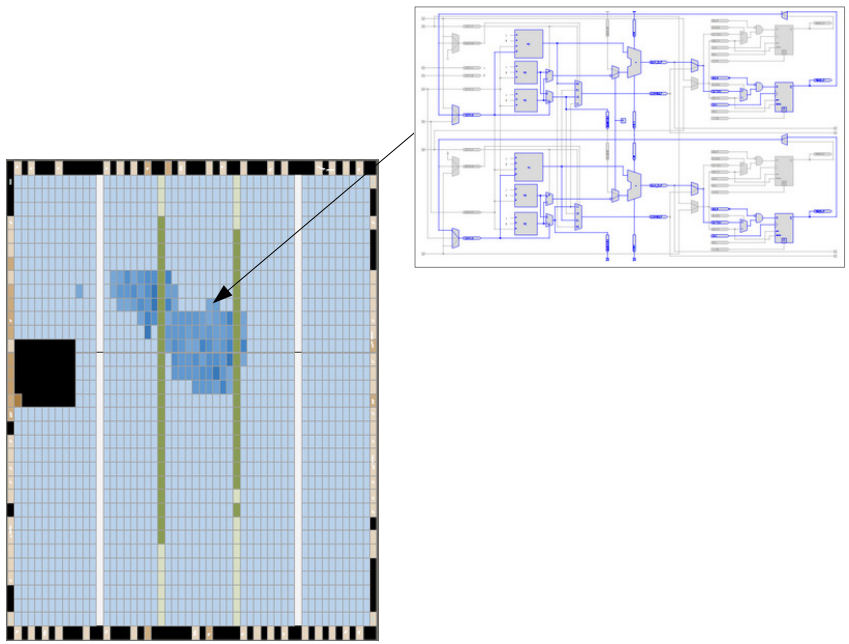


Target Machine

Power-efficiency and FPGA

FPGA = reconfigurable circuit

$\rightsquigarrow$ hope: better than GPUs (Microsoft CNN/FPGA: 53% more energy efficiency than GPU implementation).



High-level synthesis (HLS) required!

$\rightsquigarrow$ need for robust, static automatic parallelization

Outline

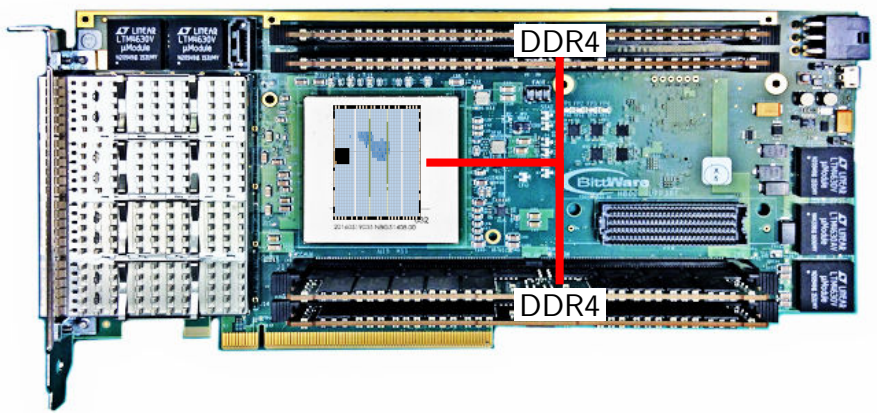
High-level synthesis for FPGA

Goal

Models and algorithms towards complete polyhedral-powered HLS for FPGA

→ XtremLogic start-up

```c
void kernel_cholesky(double **A, int N)
{
  int i, j, k;
  for (i = 0; i < N; i++) {
    for (j = 0; j < i; j++) {
      for (k = 0; k < j; k++)
        A[i][j] = A[i][j] - A[i][k] * A[j][k];
      A[i][j] = A[i][j] / A[j][j];
    }
    for (k = 0; k < i; k++)
      A[i][i] = A[i][i] - A[i][k] * A[i][k];
    A[i][i] = sqrt(A[i][i]);
  }
}
```



High-level synthesis for FPGA

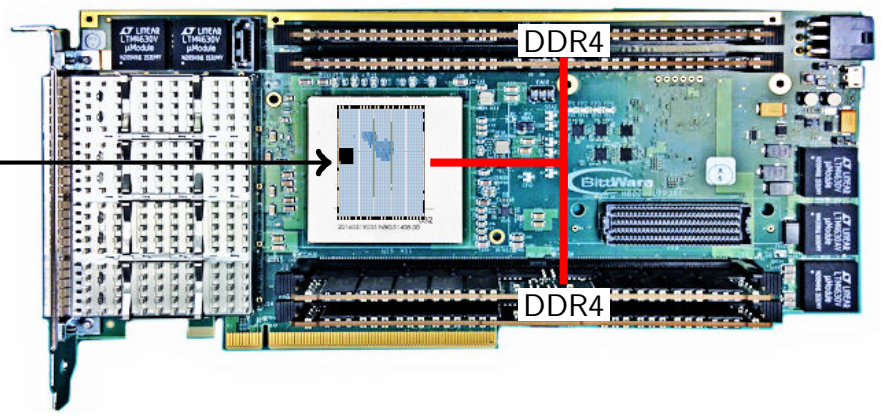
Goal

Models and algorithms towards complete polyhedral-powered HLS for FPGA

→ XtremLogic start-up

```
void kernel_cholesky(double **A, int N)
{
  int i, j, k;
  for (i = 0; i < N; i++) {
    for (j = 0; j < i; j++) {
      for (k = 0; k < j; k++)
        A[i][j] = A[i][j] - A[i][k] * A[j][k];
      A[i][j] = A[i][j] / A[j][j];
    }
    for (k = 0; k < i; k++)
      A[i][i] = A[i][i] - A[i][k] * A[i][k];
    A[i][i] = sqrt(A[i][i]);
  }
}
```

→ HLS →



High-level synthesis for FPGA

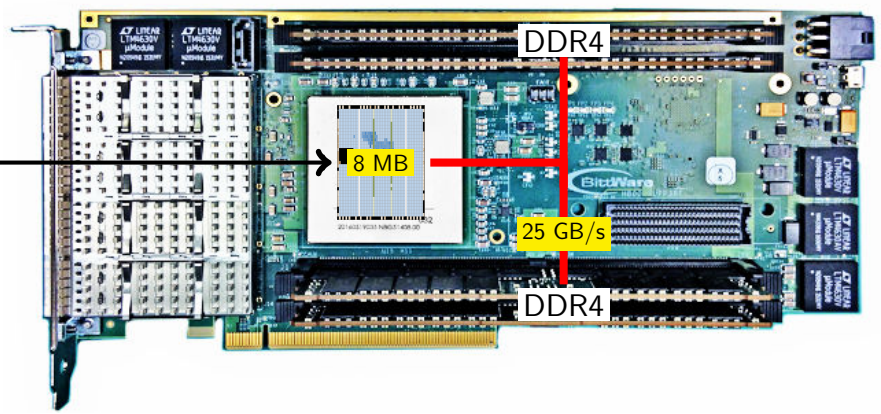
Goal

Models and algorithms towards complete polyhedral-powered HLS for FPGA

→ XtremLogic start-up

```
void kernel_cholesky(double **A, int N)
{
  int i, j, k;
  for (i = 0; i < N; i++) {
    for (j = 0; j < i; j++) {
      for (k = 0; k < j; k++)
        A[i][j] = A[i][j] - A[i][k] * A[j][k];
      A[i][j] = A[i][j] / A[j][j];
    }
    for (k = 0; k < i; k++)
      A[i][i] = A[i][i] - A[i][k] * A[i][k];
    A[i][i] = sqrt(A[i][i]);
  }
}
```

→ HLS →



High-level synthesis for FPGA

Goal

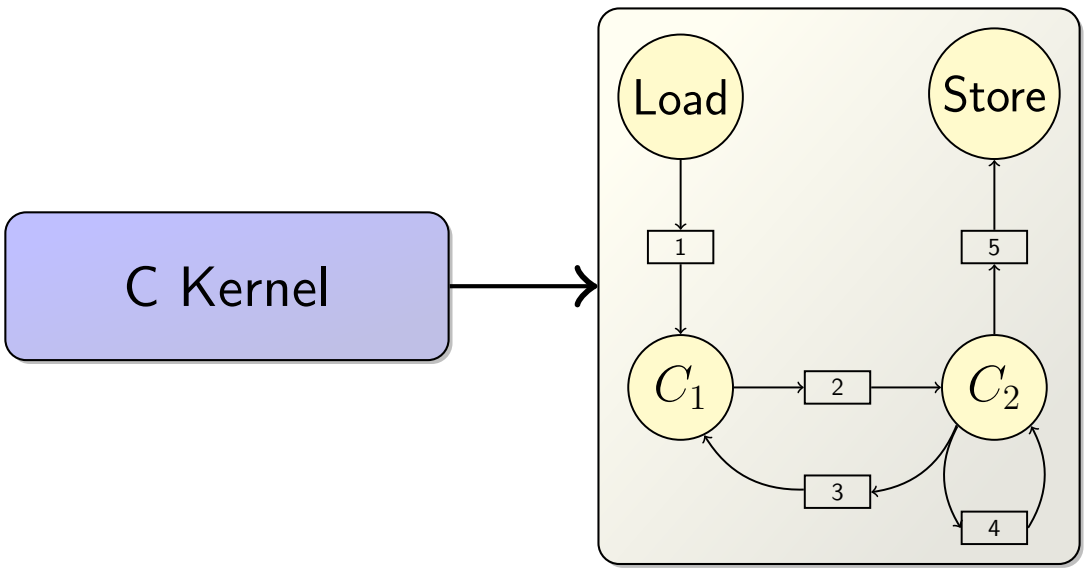
Models and algorithms towards complete polyhedral-powered HLS for FPGA → XtremLogic start-up

Challenges:

- Orchestrate data spilling to the DDR
  - Limited local storage (8 Mbytes on Stratix 10 GX1150)
  - Applications with large storage footprint
- Allow tunable operational intensity and parallelism
- Express pipelined parallelism (circuit)

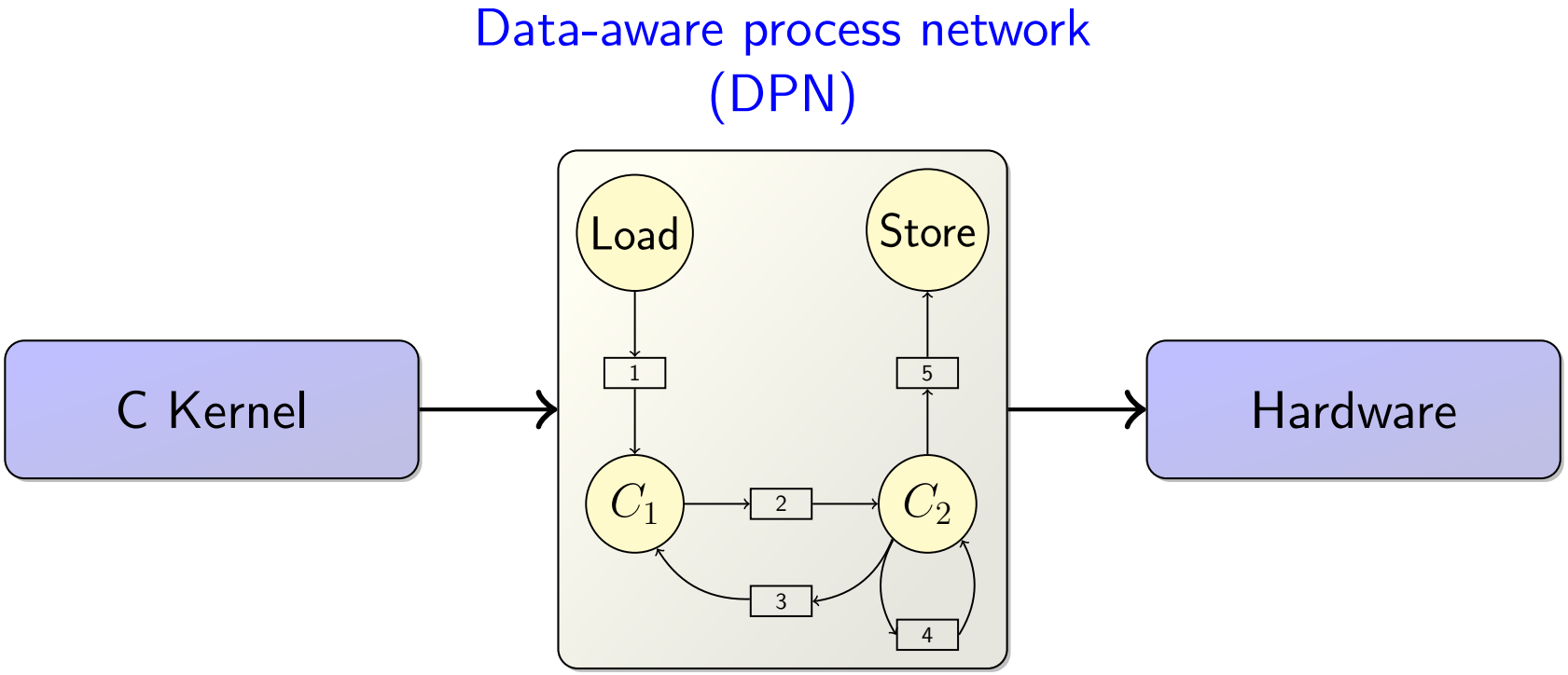
Approach

Data-aware process network (DPN)



- Produce a dataflow model with explicit data spilling
- Tunable local storage / operational intensity and parallelism

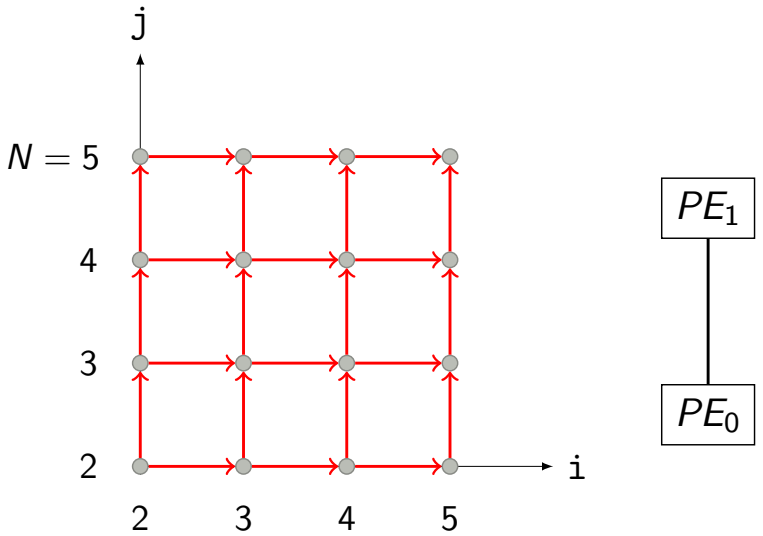
Approach



- Produce a dataflow model with explicit data spilling
- Tunable local storage / operational intensity and parallelism
- Synthesize the dataflow model to FPGA

Focus: regular loop kernels

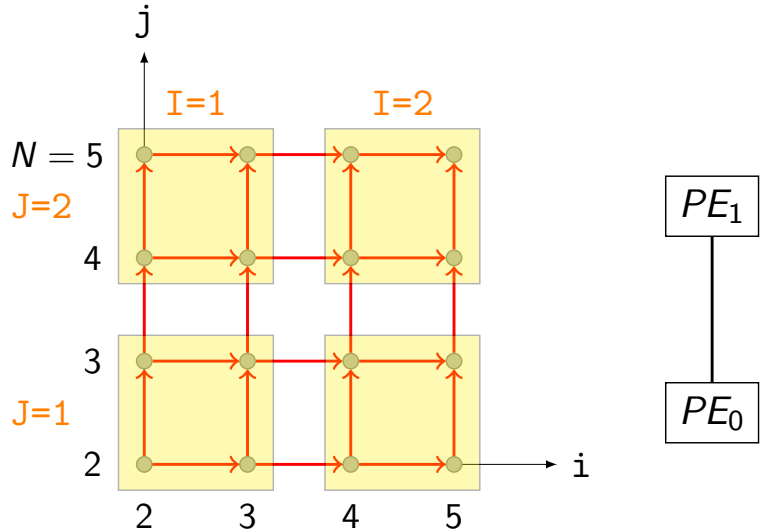
for $i := 2$ **to** N
  for $j := 2$ **to** N
    a[i,j] := a[i-1,j] + a[i,j-1];



- Polyhedral model: the all-affine world

Focus: regular loop kernels

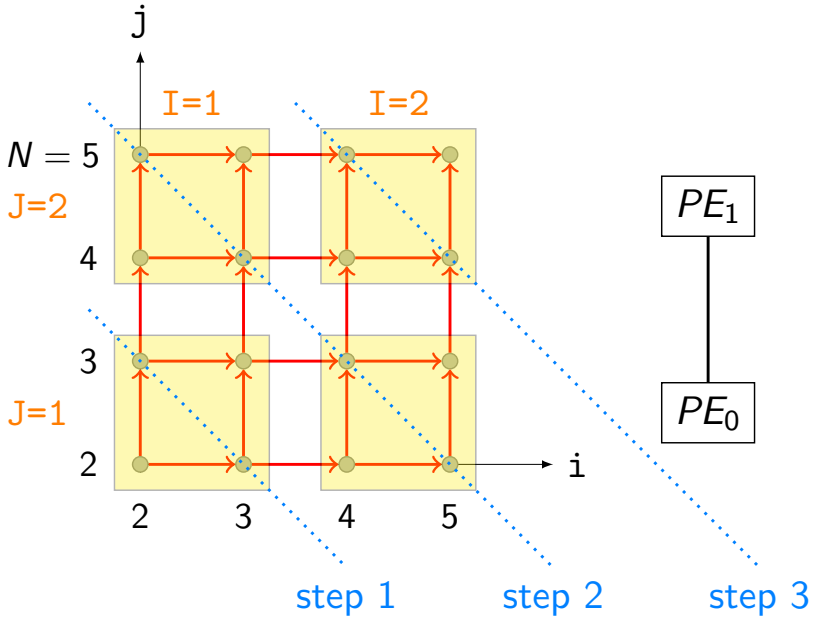
for $i := 2$ **to** N
  for $j := 2$ **to** N
    a[i,j] := a[i-1,j] + a[i,j-1];



- Polyhedral model: the all-affine world
  $\phi(i,j) = (i,j)$

Focus: regular loop kernels

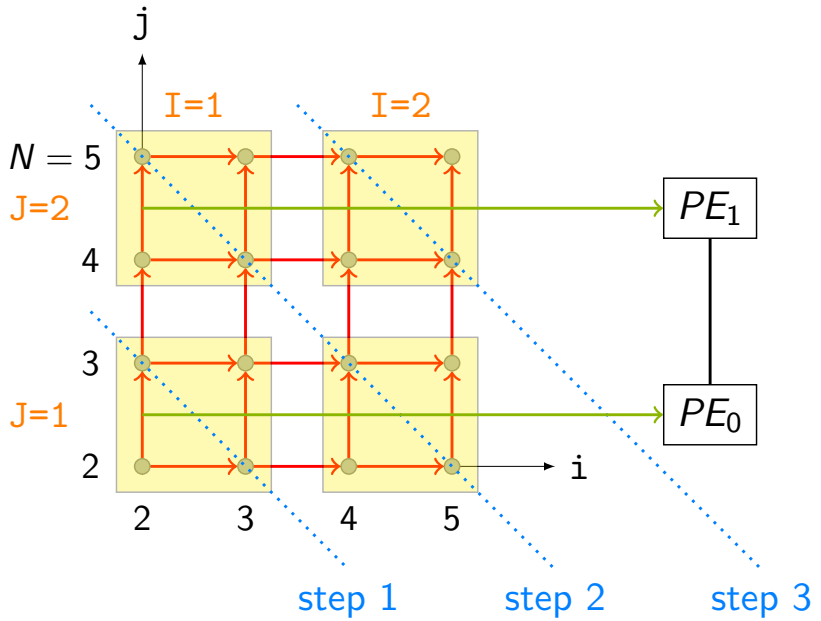
for $i := 2$ **to** N
  for $j := 2$ **to** N
    a[i,j] := a[i-1,j] + a[i,j-1];



- Polyhedral model: the all-affine world
  $\phi(i,j) = (i,j)$ $\theta(I,J,i,j) = (I+J,i,j)$

Focus: regular loop kernels

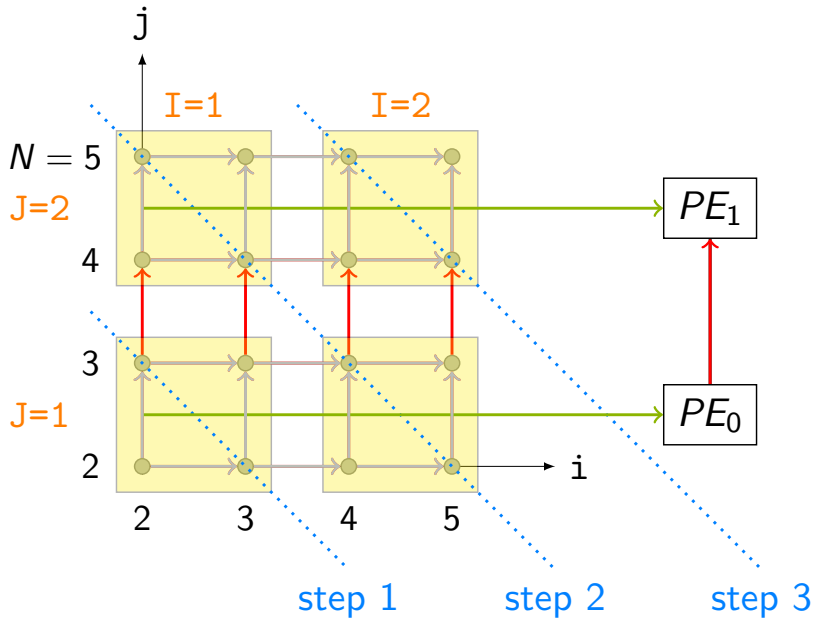
for $i := 2$ **to** N
  for $j := 2$ **to** N
    a[i,j] := a[i-1,j] + a[i,j-1];



- Polyhedral model: the all-affine world
  $\phi(i,j) = (i,j)$ $\theta(I,J,i,j) = (I+J,i,j)$ $\Pi(I,J,i,j) = J$

Focus: regular loop kernels

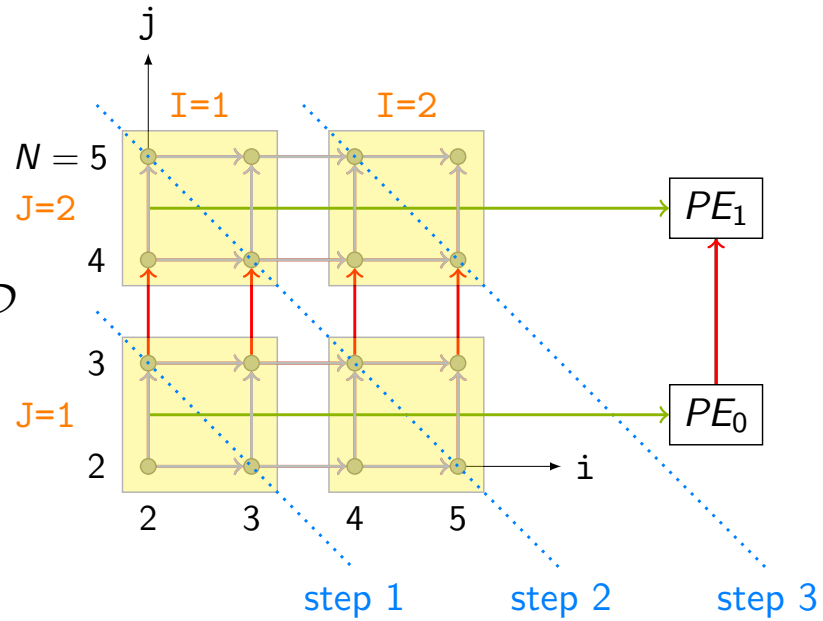
for $i := 2$ **to** N
  for $j := 2$ **to** N
    a[i,j] := a[i-1,j] + a[i,j-1];



- Polyhedral model: the all-affine world
  $\phi(i,j) = (i,j)$ $\theta(I,J,i,j) = (I+J,i,j)$ $\Pi(I,J,i,j) = J$

Focus: regular loop kernels

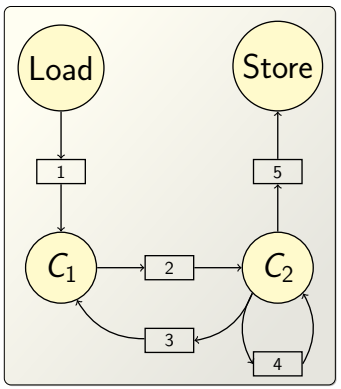
$$a[i,j] = a[i-1,j] + a[i,j-1] \quad \forall (i,j) \in \mathcal{D}$$



- Polyhedral model: the all-affine world
  $\phi(i,j) = (i,j)$ $\theta(I,J,i,j) = (I+J,i,j)$ $\Pi(I,J,i,j) = J$
- Polyhedral representation: System of affine recurrence equations (SARE) + Reductions ($\sum_k$)

Contributions

Data-aware process networks (DPN) [Patent14]

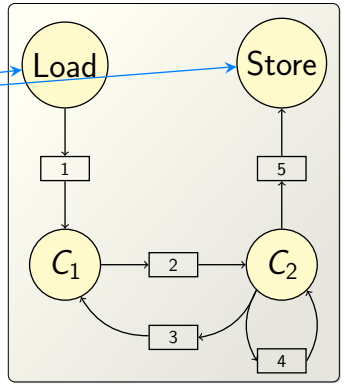


Contributions

Data-aware process networks (DPN) [Patent14]

Communication synthesis

[ASAP10,PPoPP12,IMPACT12,DATE13]



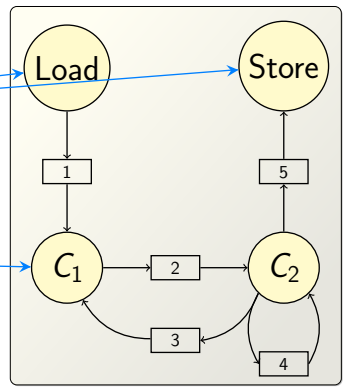
Contributions

Data-aware process networks (DPN) [Patent14]

Communication synthesis

[ASAP10,PPoPP12,IMPACT12,DATE13]

- Control scheduling [ARC11,MICPRO12]
- Control synthesis [TACO17]



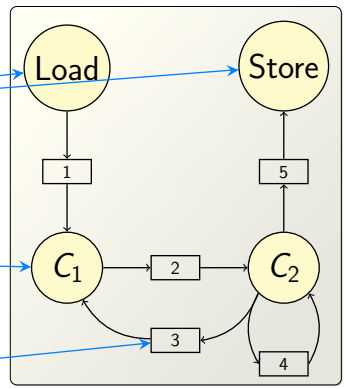
Contributions

Data-aware process networks (DPN) [Patent14]

Communication synthesis
[ASAP10,PPoPP12,IMPACT12,DATE13]

- Control scheduling [ARC11,MICPRO12]
- Control synthesis [TACO17]

- Channel typing [HiP3ES18]
- Channel allocation/sizing [LCTES07]
- Synchronizations [Patent14]



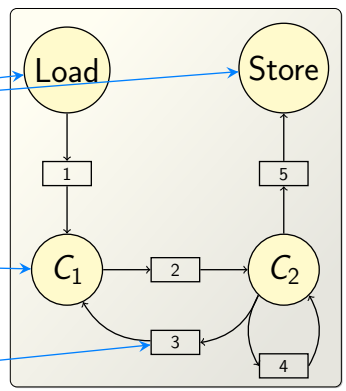
Contributions

Data-aware process networks (DPN) [Patent14]

Communication synthesis
[ASAP10,PPoPP12,IMPACT12,DATE13]

- Control scheduling [ARC11,MICPRO12]
- Control synthesis [TACO17]

- Channel typing [HiP3ES18]
- Channel allocation/sizing [LCTES07]
- Synchronizations [Patent14]



Software: dcc (DPN C Compiler)
Transferred to XtremLogic under an Inria license

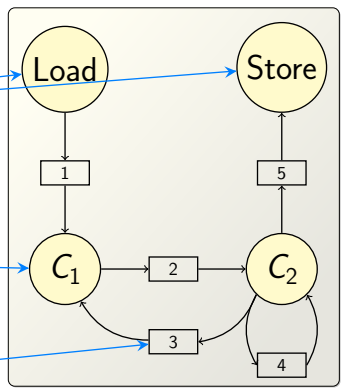
Contributions

Data-aware process networks (DPN) [Patent14]

Communication synthesis
[ASAP10,PPoPP12,IMPACT12,DATE13]

- Control scheduling [ARC11,MICPRO12]
- **Control synthesis** [TACO17]

- **FIFO recovery** [HiP3ES18]
- Channel allocation/sizing [LCTES07]
- Synchronizations [Patent14]



Software: `dcc` (DPN C Compiler)
Transferred to XtremLogic under an Inria license

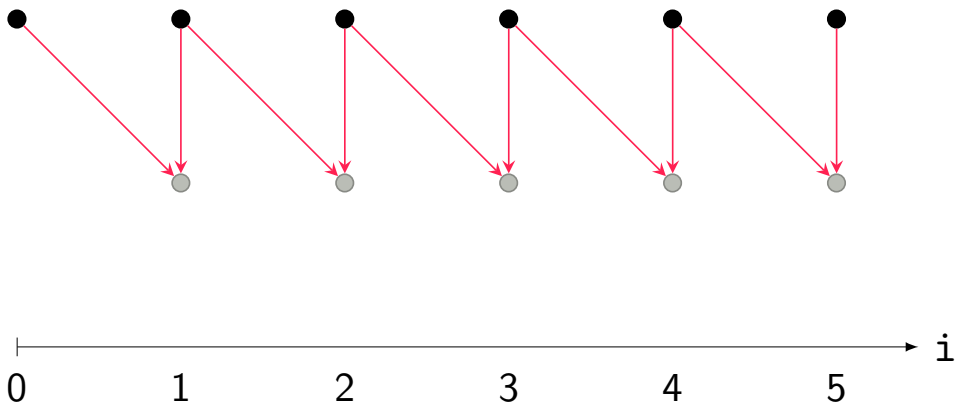
Regular process networks

for $i := 0$ **to** N

- $a[i] = f(i);$

for $i := 1$ **to** N

- $b[i] := a[i-1] + a[i];$



- Partition of the computation: processes
- Partition of $\rightarrow_{pc}$: channels $\{\rightarrow_1, \rightarrow_2, \ldots\}$
- A schedule θ_P for each process P

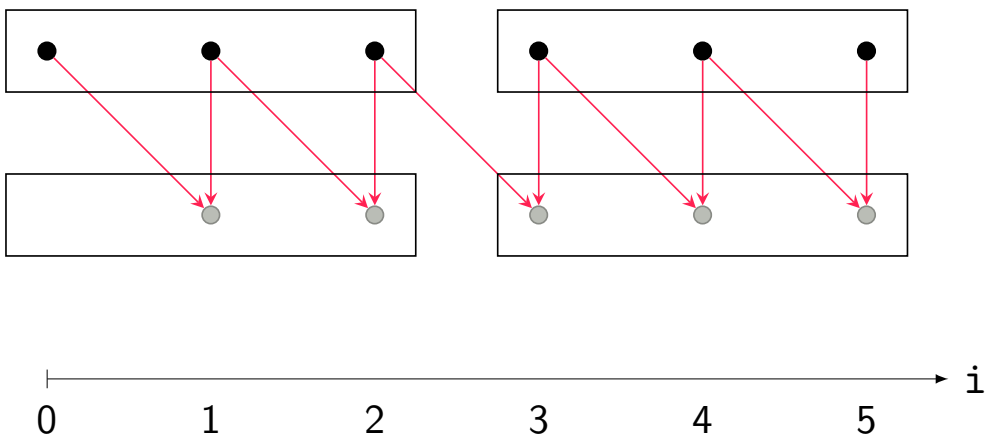
Regular process networks

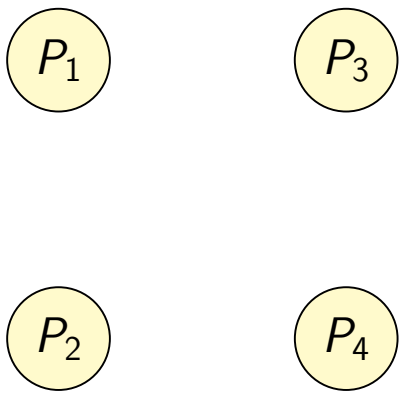
for $i := 0$ **to** N

- $a[i] = f(i);$

for $i := 1$ **to** N

- $b[i] := a[i-1] + a[i];$





- Partition of the computation: processes
- Partition of $\rightarrow_{pc}$: channels $\{\rightarrow_1, \rightarrow_2, \ldots\}$
- A schedule θ_P for each process P

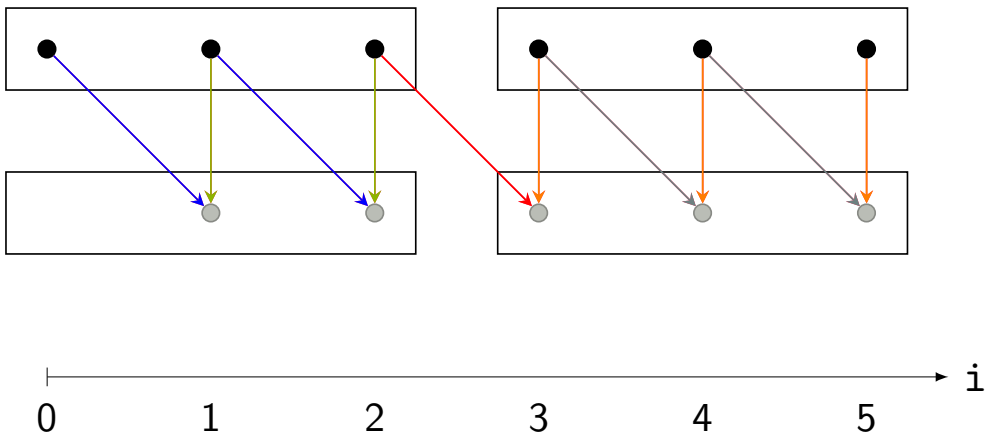
Regular process networks

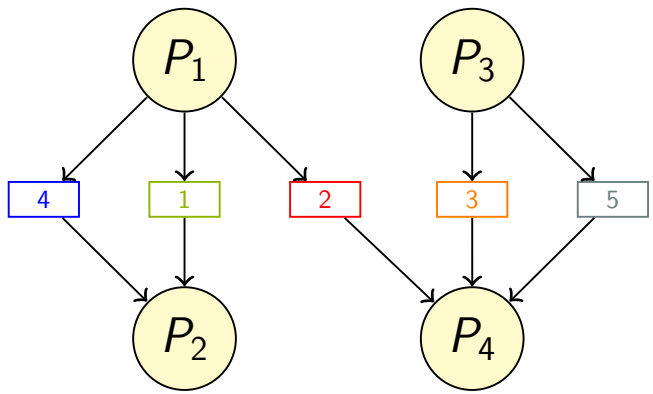
for $i := 0$ **to** N

- $a[i] = f(i);$

for $i := 1$ **to** N

- $b[i] := a[i-1] + a[i];$





- Partition of the computation: processes
- Partition of $\rightarrow_{pc}$: channels $\{\rightarrow_1, \rightarrow_2, \ldots\}$
- A schedule θ_P for each process P

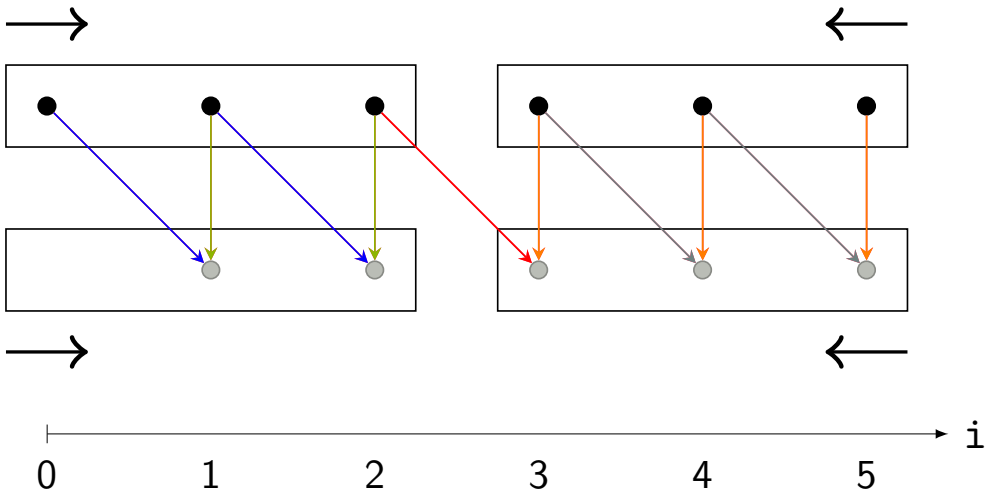
Regular process networks

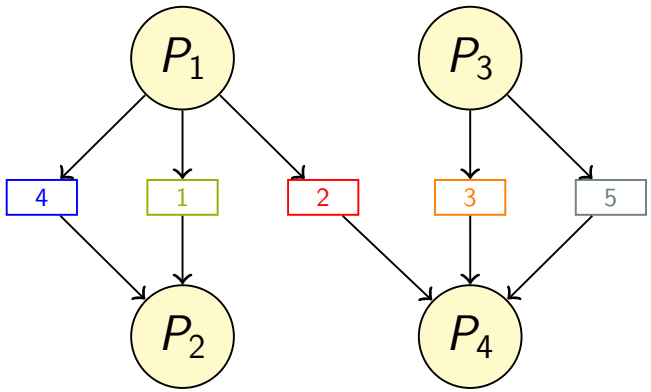
for $i := 0$ **to** N

- $a[i] = f(i);$

for $i := 1$ **to** N

- $b[i] := a[i-1] + a[i];$





- Partition of the computation: processes
- Partition of $\rightarrow_{pc}$: channels $\{\rightarrow_1, \rightarrow_2, \ldots\}$
- A schedule θ_P for each process P

Regular process networks

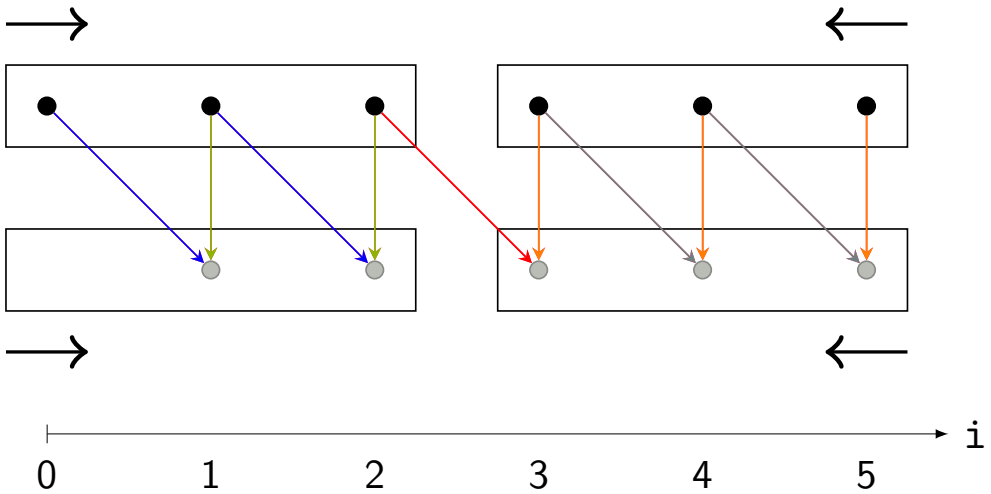
for $i := 0$ **to** N

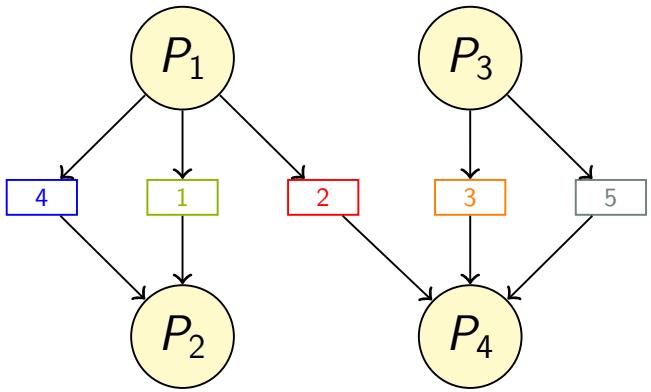
- $a[i] = f(i);$

for $i := 1$ **to** N

- $b[i] := a[i-1] + a[i];$

Locally sequential
Globally dataflow





- Partition of the computation: processes
- Partition of $\rightarrow_{pc}$: channels $\{\rightarrow_1, \rightarrow_2, \ldots\}$
- A schedule θ_P for each process P

HLS methodology

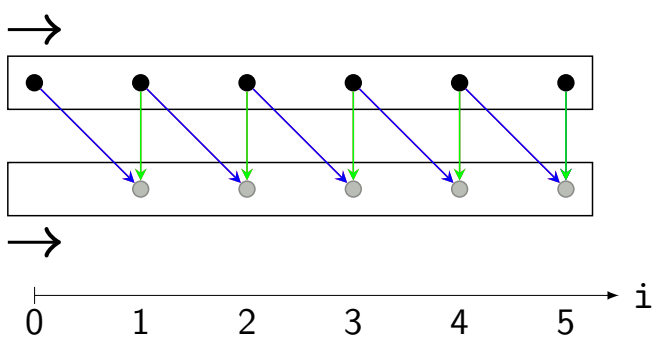
1. Setup the RPN partitionning strategy (e.g. PPN, DPN)
2. Front-end: kernel $\longrightarrow$ RPN
3. Back-end: RPN $\longrightarrow$ circuit

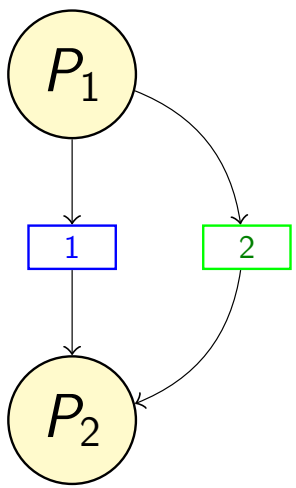
Benefits:

- Combines the benefits of partionning and dataflow models
- Explicit and typed communications (FIFO, buffer, DDR)

for $i := 0$ **to** N
- $a[i] = f(i)$;

for $i := 1$ **to** N
- $b[i] := a[i-1] + a[i]$;

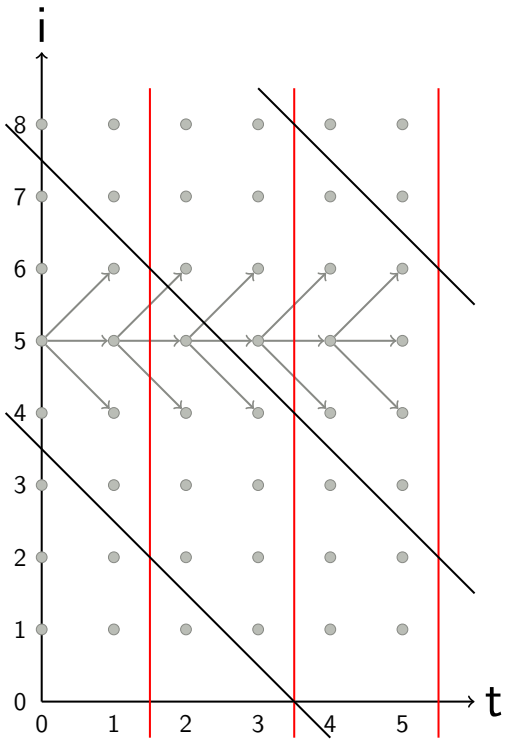




Data-aware process networks [Patent'14]

for $t := 1$ **to** T
  for $i := 1$ **to** $N - 2$
    $a[t, i] := a[t-1, i-1] +$
      $a[t-1, i] +$
      $a[t-1, i+1];$

Tiling $\phi_S(t, i) = (t, t + i)$

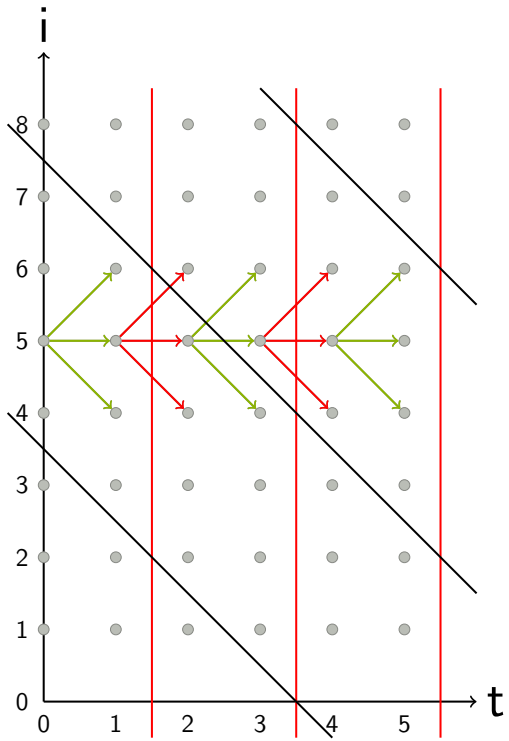


- Load/store: consider tile bands as reuse units [ASAP10,DATE13]
  $\rightsquigarrow$Pipelined comm.: Load(T) $\rightarrow$ C(T) $\rightarrow$ Store(T)

Data-aware process networks [Patent'14]

for $t := 1$ **to** T
  for $i := 1$ **to** $N - 2$
    $a[t, i] := a[t-1, i-1] +$
      $a[t-1, i] +$
      $a[t-1, i+1];$

Tiling $\phi_S(t, i) = (t, t + i)$

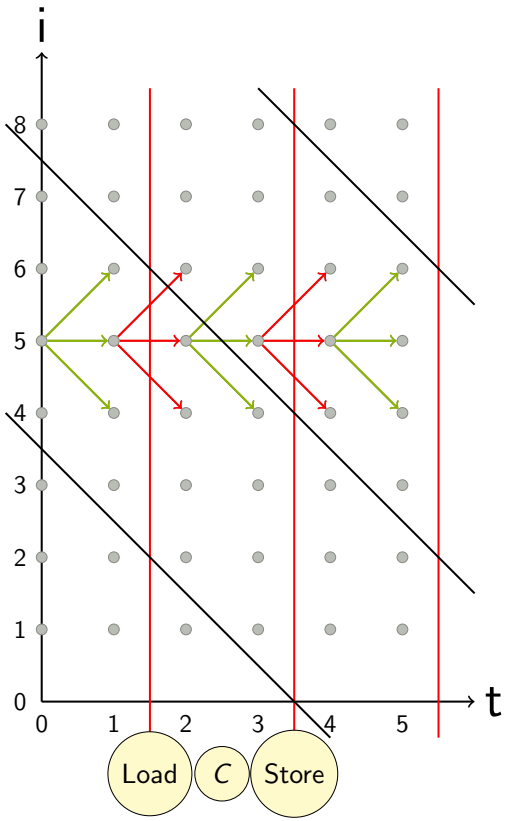


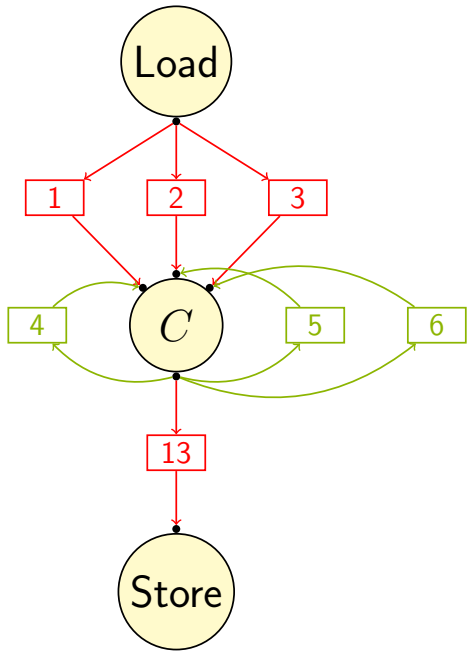
- Load/store: consider tile bands as reuse units [ASAP10,DATE13]
  $\rightsquigarrow$Pipelined comm.: Load(T) $\rightarrow$ C(T) $\rightarrow$ Store(T)

Data-aware process networks [Patent'14]

for $t := 1$ **to** T
  for $i := 1$ **to** $N - 2$
    $a[t, i] := a[t-1, i-1] +$
      $a[t-1, i] +$
      $a[t-1, i+1];$

Tiling $\phi_S(t, i) = (t, t + i)$



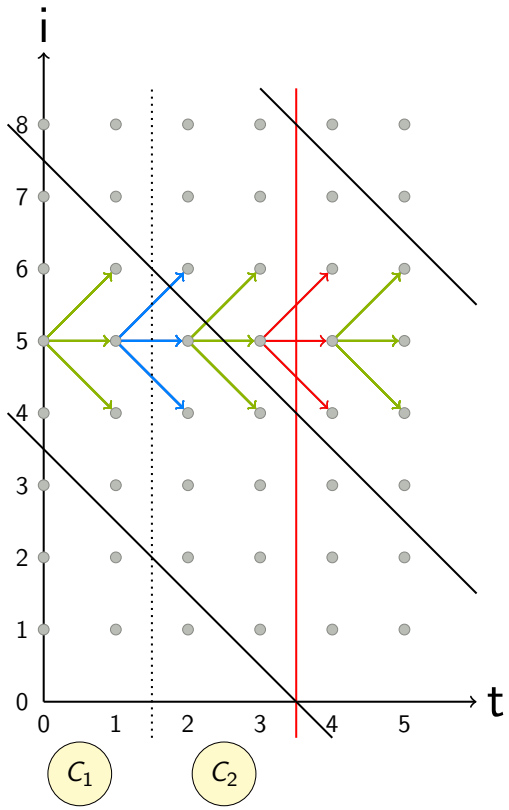


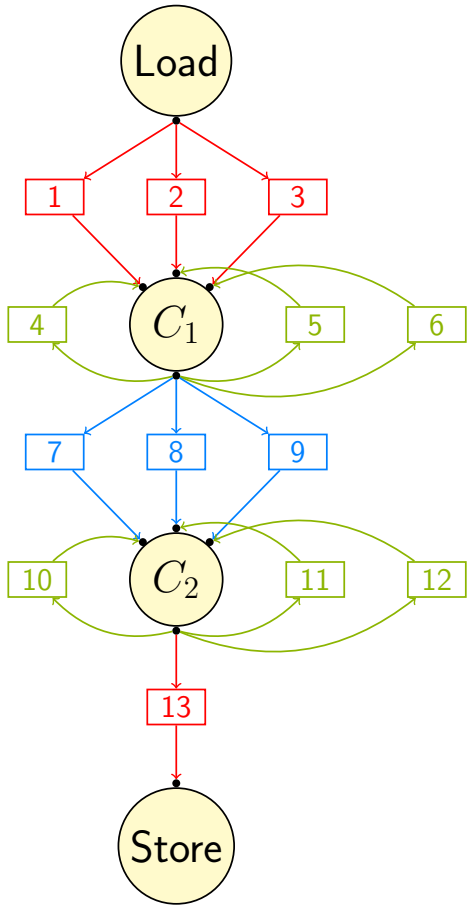
- Load/store: consider tile bands as reuse units [ASAP10,DATE13]
  $\rightsquigarrow$Pipelined comm.: Load(T) $\rightarrow$ C(T) $\rightarrow$ Store(T)

Data-aware process networks [Patent'14]

for $t := 1$ **to** T
  for $i := 1$ **to** $N - 2$
    $a[t, i] := a[t-1, i-1] +$
      $a[t-1, i] +$
      $a[t-1, i+1];$

Tiling $\phi_S(t, i) = (t, t + i)$





- Load/store: consider tile bands as reuse units [ASAP10,DATE13]
  $\rightsquigarrow$Pipelined comm.: Load(T) $\rightarrow$ C(T) $\rightarrow$ Store(T)
- Parallelism: split tile band with outer tiling hyperplanes (ex: t)

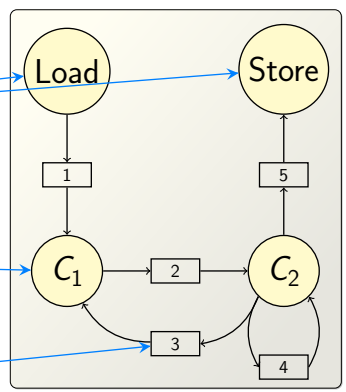
Outline

Data-aware process networks (DPN) [Patent14]

Communication synthesis
[ASAP10,PPoPP12,IMPACT12,DATE13]

- Control scheduling [ARC11,MICPRO12]
- **Control synthesis** [TACO17]

- **FIFO recovery** [HiP3ES18]
- Channel allocation/sizing [LCTES07]
- Synchronizations [Patent14]



Software: `dcc` (DPN C Compiler)
Transferred to XtremLogic under an Inria license

Channel restructuring for FIFO recovery [HiP3ES'18]

Goals

- Compile DPN channels
- Preferably with FIFO
  $\rightsquigarrow$ light silicon surface, less synchronization overhead

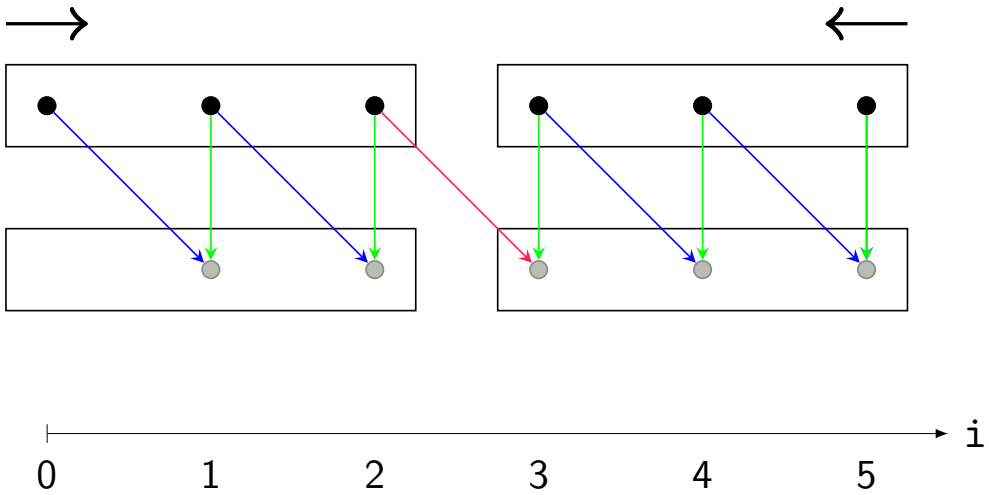
Challenge

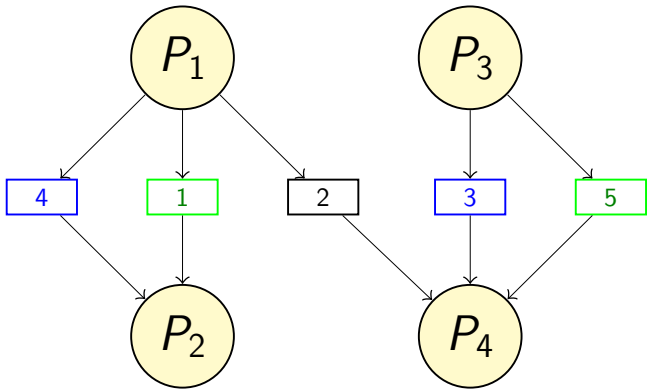
DPN relies on loop tiling, which breaks most FIFO channels

Approach

- Restructure the channels so most FIFO are recovered
- Theorem: the recovery is complete on DPN

Communication Patterns

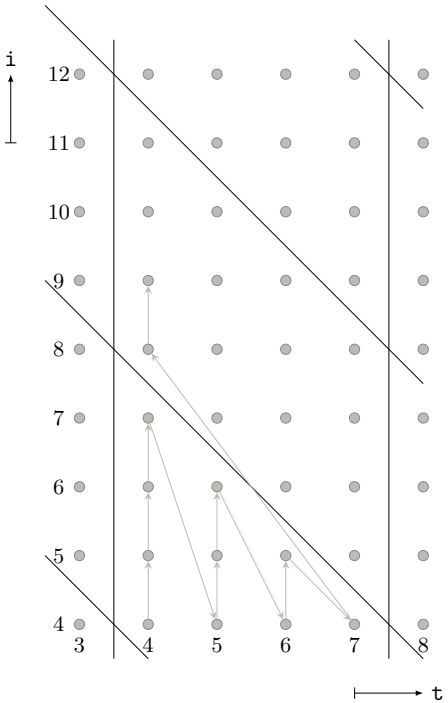


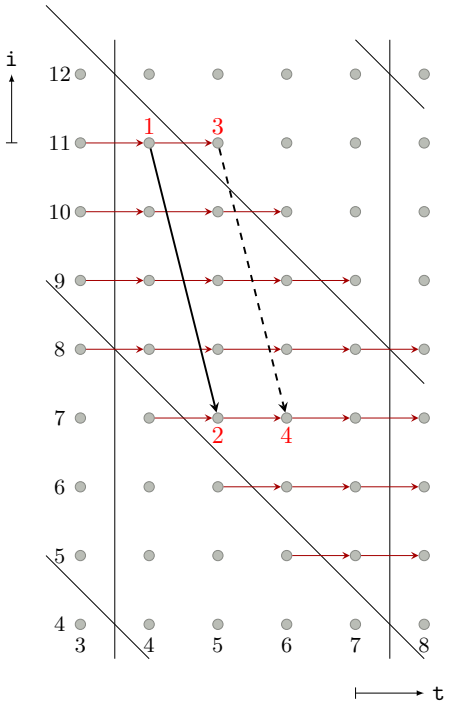


A channel might be implemented by a FIFO iff

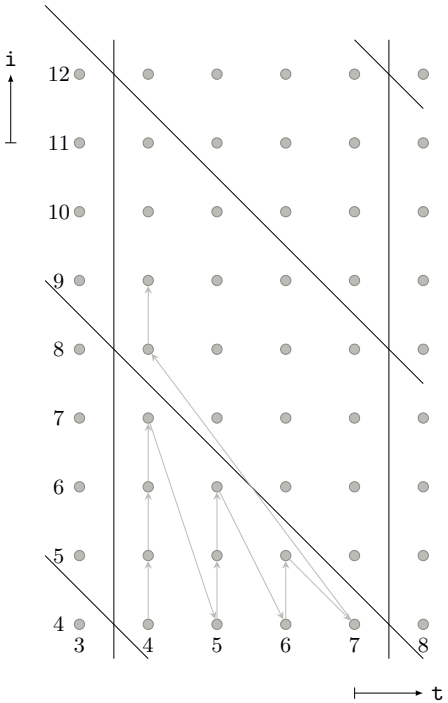
- the values are read in the production order (*in-order*)
- each value is read exactly once (*unicity*)

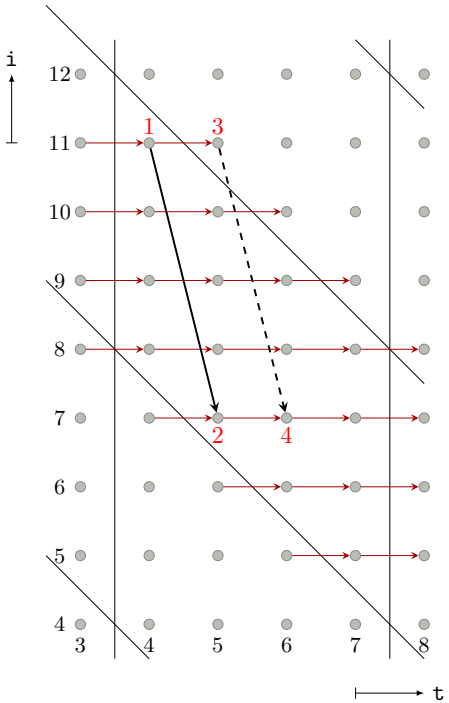
How loop tiling breaks FIFO channels

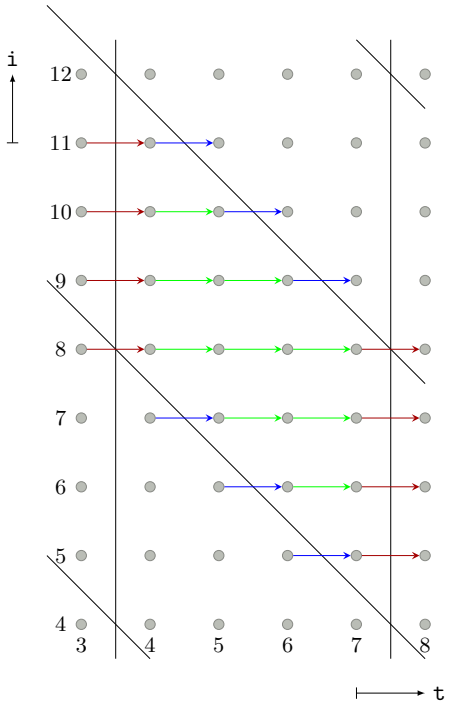




How loop tiling breaks FIFO channels







Solution: channel restructuring

Experimental evaluation

Kernel	PPN	PPN with tiling			DPN		
	#fifo	#rem	#rec	%	#rem	#rec	%
trmm	2	1	1	100	1	1	100
gemm	2	1	1	100	1	1	100
syrk	2	1	1	100	1	1	100
symm	6	3	3	100	5	1	100
gemver	4	2	2	100	3	1	100
gesummv	6	6	0	100	6	0	100
syr2k	2	1	1	100	1	1	100
lu	3	0	3	100	0	3	100
trisolv	4	3	1	100	3	1	100
cholesky	6	3	3	100	4	2	100
doitgen	3	2	1	100	2	1	100
bicg	4	2	2	100	2	2	100
mvt	2	0	2	100	0	2	100
3mm	6	2	2	50	3	3	100
2mm	4	2	1	50	2	2	100
covariance	7	4	2	66	4	3	100
correlation	13	9	3	75	9	4	100
fdtd-2d	12	0	6	50	5	7	100
jacobi-2d	10	0	2	20	2	8	100
seidel-2d	9	0	3	33	2	7	100
jacobi-1d	6	1	5	100	2	4	100
heat-3d	20	0	0	0	2	18	100

- PolyBench/C v3.2 kernels
- Rsults on PPN with DPN partitioning (DPN)
- Results on PPN, without DPN partitioning (PPN with tiling).

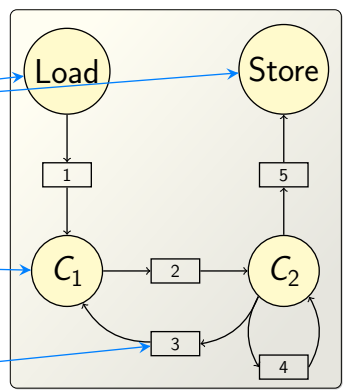
Outline

Data-aware process networks (DPN) [Patent14]

Communication synthesis
[ASAP10,PPoPP12,IMPACT12,DATE13]

- Control scheduling [ARC11,MICPRO12]
- **Control synthesis** [TACO17]

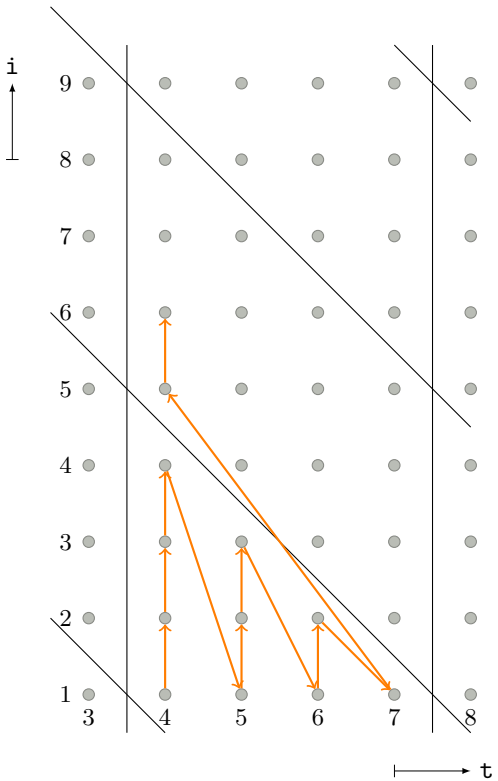
- **FIFO recovery** [HiP3ES18]
- Channel allocation/sizing [LCTES07]
- Synchronizations [Patent14]



Software: `dcc` (DPN C Compiler)
Transferred to XtremLogic under an Inria license

Control synthesis [TACO'17]

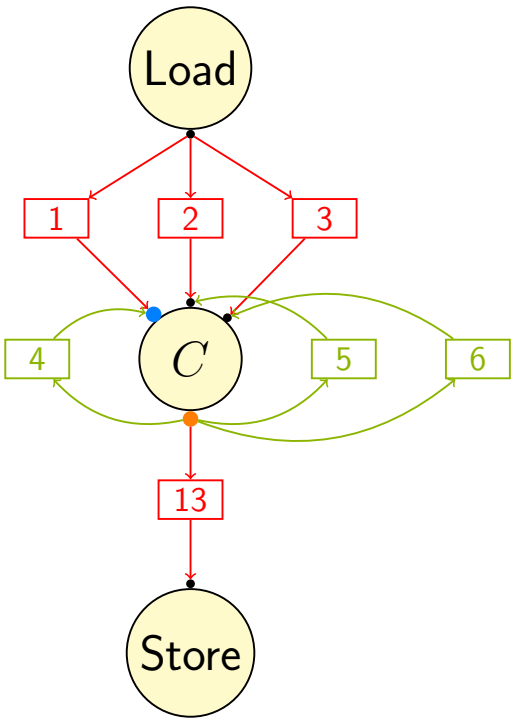
Finite state machine



$$\text{First}_P = \{\ T \geq 1 \wedge N \geq 3 : \quad (0,0,1,1)$$

$$\text{Next}_P(l_1, l_2, t, i) = \begin{cases}
\text{(depth } l_1\text{)} \\
-4 + T - 4l_1 \geq 0 : \\
\quad (1 + l_1, 1 + l_1, 4 + 4l_1, 1) \\
\\
\text{(depth } l_2\text{)} \\
-3 + N + 4l_1 - 4l_2 \geq 0 \wedge 6 - N - 4l_1 + 4l_2 \geq 0 \wedge \\
-6 + N + T - 4l_2 \geq 0 \wedge -5 + N - 4l_2 < 0 : \\
\quad (l_1, 1 + l_2, 6 - N + 4l_2, -2 + N) \\
\\
l_1 > 0 \wedge -5 + N - 4l_2 \geq 0 \\
\quad (l_1, 1 + l_2, 4l_1, 4 - 4l_1 + 4l_2) \\
\\
6 - N - 4l_1 + 4l_2 < 0 \wedge -6 + N + T - 4l_2 \geq 0 \wedge -5 + N - 4l_2 < 0 \\
\quad (l_1, 1 + l_2, 4l_1, 4 - 4l_1 + 4l_2) \\
\\
-l_1 \geq 0 \wedge -5 + N - 4l_2 \geq 0 \\
\quad (l_1, 1 + l_2, 1, 3 + 4l_2) \\
\\
\text{(depth } t\text{)} \\
2 + t - 4l_2 \geq 0 \wedge 1 - t + 4l_2 \geq 0 \wedge \\
2 - t + 4l_1 \geq 0 \wedge -1 + T - t \geq 0 \\
\quad (l_1, l_2, 1 + t, 1) \\
\\
2 + t - 4l_2 < 0 \wedge 1 - t + 4l_2 \geq 0 \wedge \\
2 - t + 4l_1 \geq 0 \wedge -1 + T - t \geq 0 \\
\quad (l_1, l_2, 1 + t, -1 - t + 4l_2) \\
\\
\text{(depth } i\text{)} \\
-3 + N - i \geq 0 \wedge 2 - t - i + 4l_2 \geq 0 \\
\quad (l_1, l_2, t, 1 + i)
\end{cases}$$

Steering logic



$$\textbf{mux}(\langle C, I_1, I_2, t, i\rangle, 1) = \begin{cases} t = 4I_1 : \\ \quad \text{buffer1}[i-1] \\ t > 4I_1 \wedge i = 1 : \\ \quad \text{buffer1}[i-1] \\ t > 4I_1 \wedge i > 1 : \\ \quad \text{buffer4}[t-1, i-1] \end{cases}$$

$$\textbf{demux}(\langle C, I_1, I_2, t, i\rangle) = \begin{cases} i \leq N-3 \wedge t < 4I_1 + 3 \wedge t \leq T-1 : \\ \quad \text{buffer4}[t, i] \\ i \leq N-2 \wedge t < 4I_1 + 3 \wedge t \leq T-1 : \\ \quad \text{buffer5}[t, i] \\ i \leq N-2 \wedge t < 4I_1 + 3 \wedge t \leq T-1 : \\ \quad \text{buffer6}[t, i] \\ t = 4I_1 + 3 : \\ \quad \text{buffer13}[i] \end{cases}$$

Approach

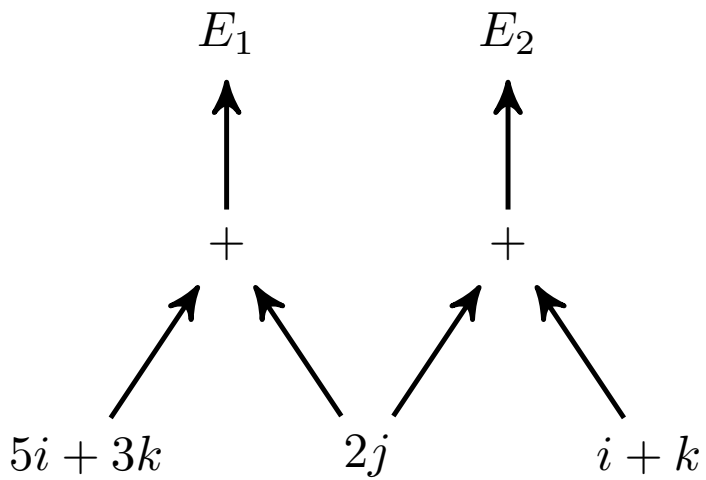
Compact the affine expressions/constraints to an hardware-efficient DAG

Challenges:

- Volume of affine expressions and constraints
- Must be evaluated in parallel
- Common subexpression factorization is not sufficient

Expression factorization

$E_1 = i + 2j + k$
$E_2 = 5i + 2j + 3k$

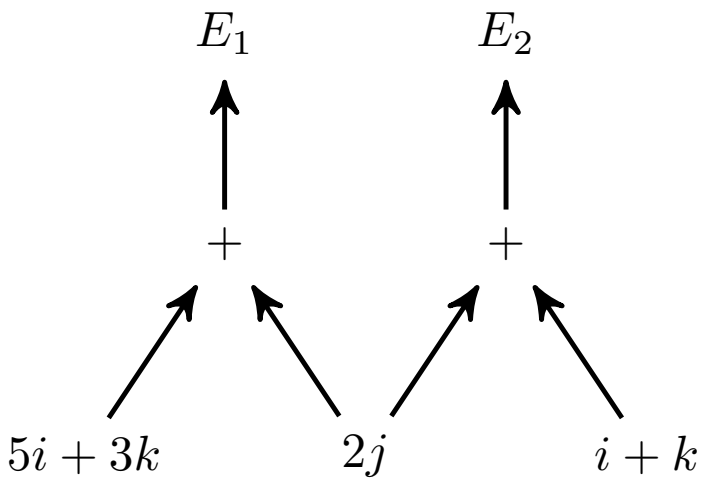


	+	× constant	shift
CSE	4	2	1

Expression factorization

$E_1 = i + 2j + k$
$E_2 = 5i + 2j + 3k$

remark that $E_2 = E_1 + 4i + 2k$

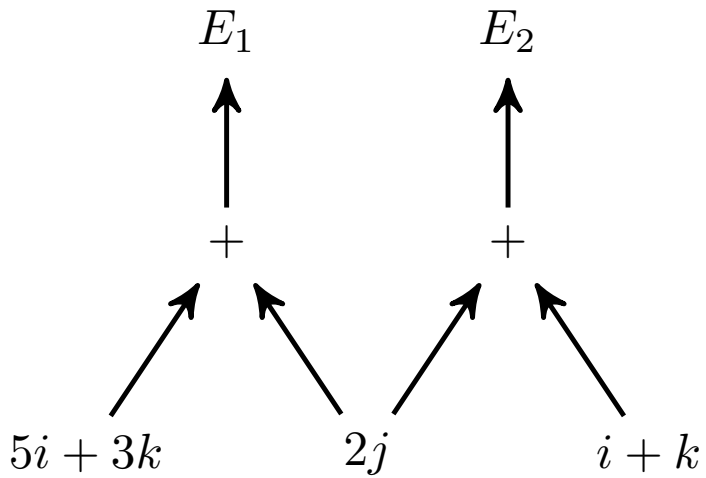


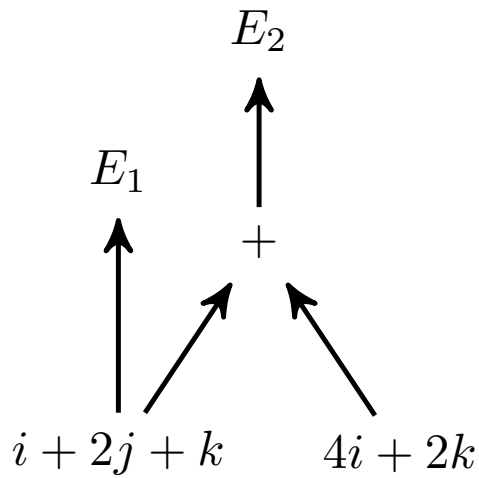
	+	× constant	shift
CSE	4	2	1

Expression factorization

$E_1 = i + 2j + k$
$E_2 = 5i + 2j + 3k$

remark that $E_2 = E_1 + 4i + 2k$

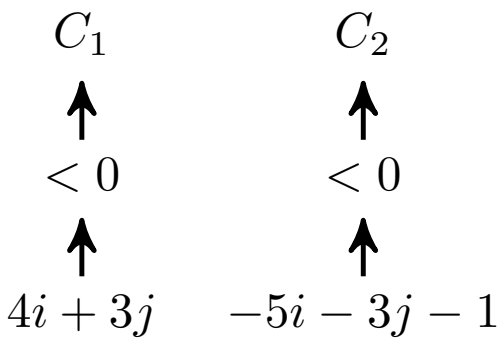




	+	× constant	shift
CSE	4	2	1
OPT	4	0	3

Constraint factorization

$C_1 : 4i + 3j < 0$
$C_2 : -5i - 3j - 1 < 0$

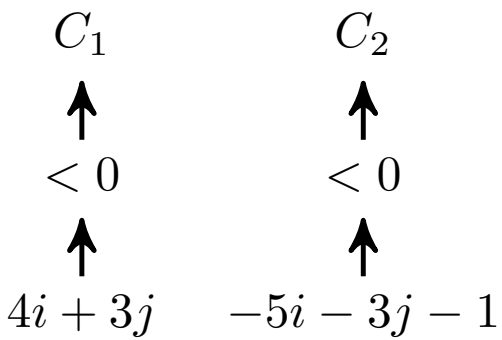


	+	× constant	shift
CSE	4	3	1

Constraint factorization

$C_1 : 4i + 3j < 0$
$C_2 : -5i - 3j - 1 < 0$

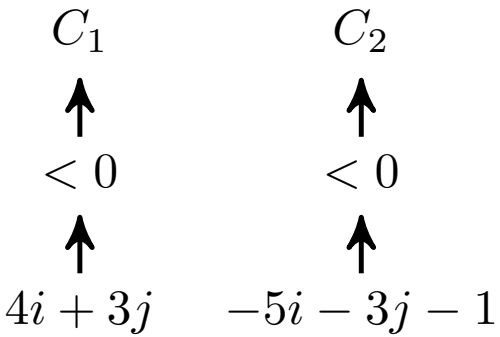
remark that: $C_2 : 5i + 3j \geq 0$



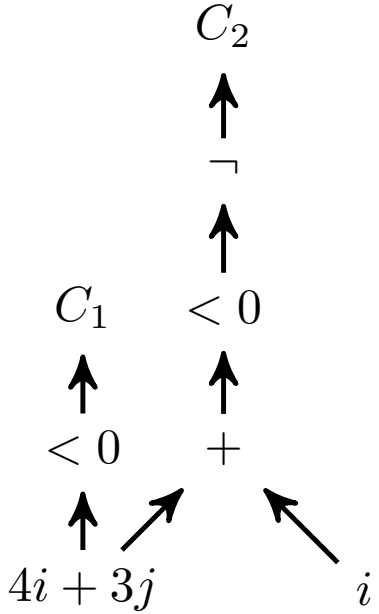
	+	× constant	shift
CSE	4	3	1

Constraint factorization

$C_1 : 4i + 3j < 0$
$C_2 : -5i - 3j - 1 < 0$



remark that: $C_2 : 5i + 3j \geq 0$



	$+$	$\times$ constant	shift
CSE	4	3	1
OPT	2	1	1

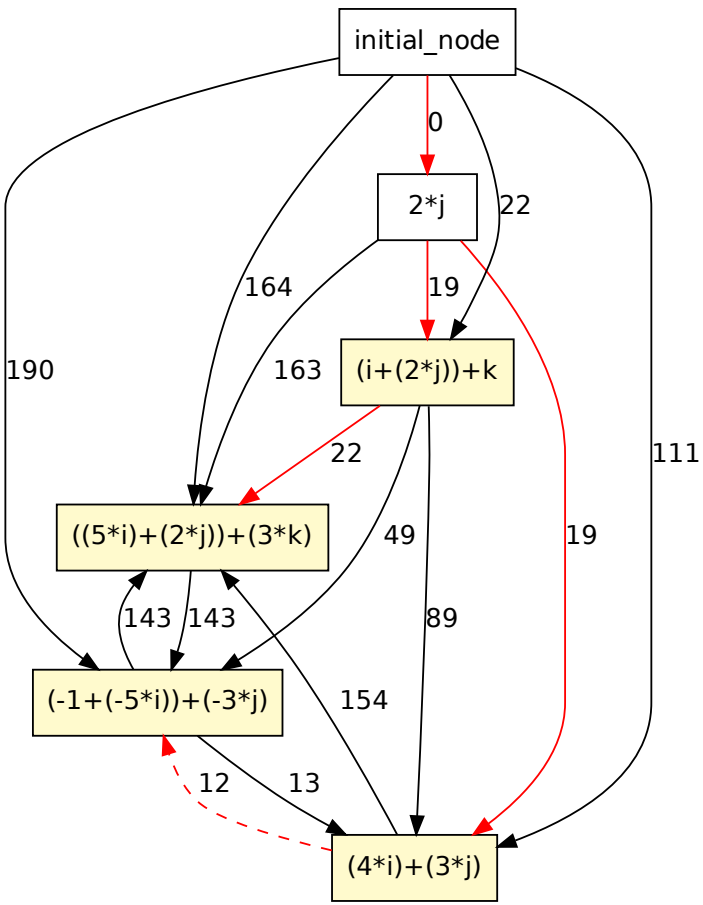
Realization graph

Nodes:
expressions, constraints, CSE

Edges: $u \xrightarrow{\Delta} v$:
v realized from u at cost Δ

$\mathcal{C}$:

Id	Constraint
1	$i + 2j + k < 0$
2	$5i + 2j + 3k < 0$
3	$4i + 3j < 0$
4	$-5i - 3j - 1 < 0$



Realization graph

Nodes:
expressions, constraints, CSE

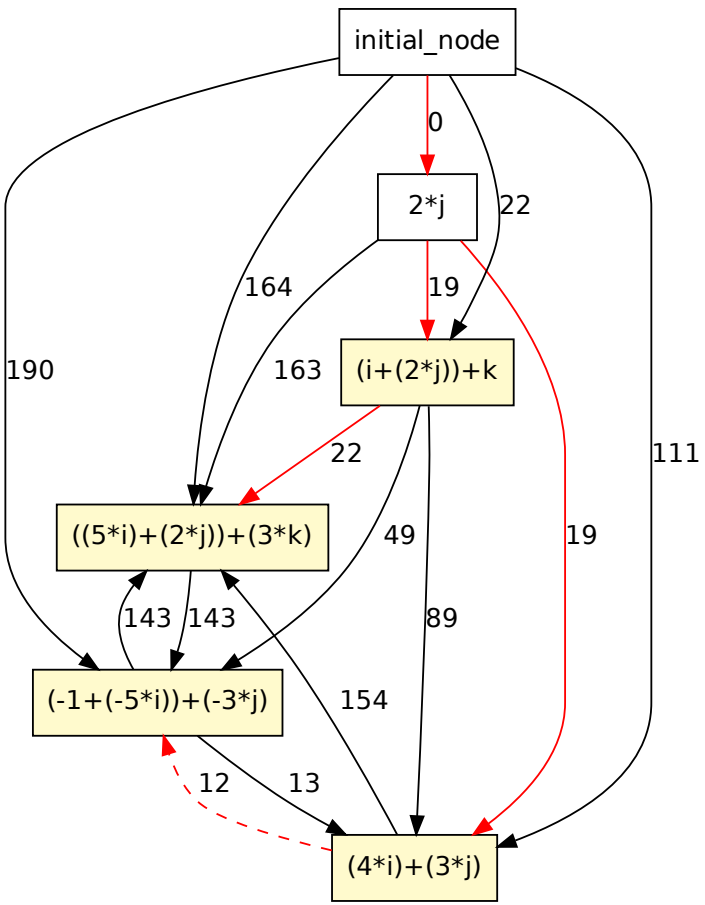
$\mathcal{C}$:

Id	Constraint
1	$i + 2j + k < 0$
2	$5i + 2j + 3k < 0$
3	$4i + 3j < 0$
4	$-5i - 3j - 1 < 0$

Edges: $u \xrightarrow{\Delta} v$:
v realized from u at cost Δ

Realization of v:
$initial\_node \xrightarrow{\Delta_1} u_1 \ldots \xrightarrow{\Delta_n} u_n \xrightarrow{\Delta} v$

Global realization:
spanning tree



Realization graph

Nodes:
expressions, constraints, CSE

$\mathcal{C}$:

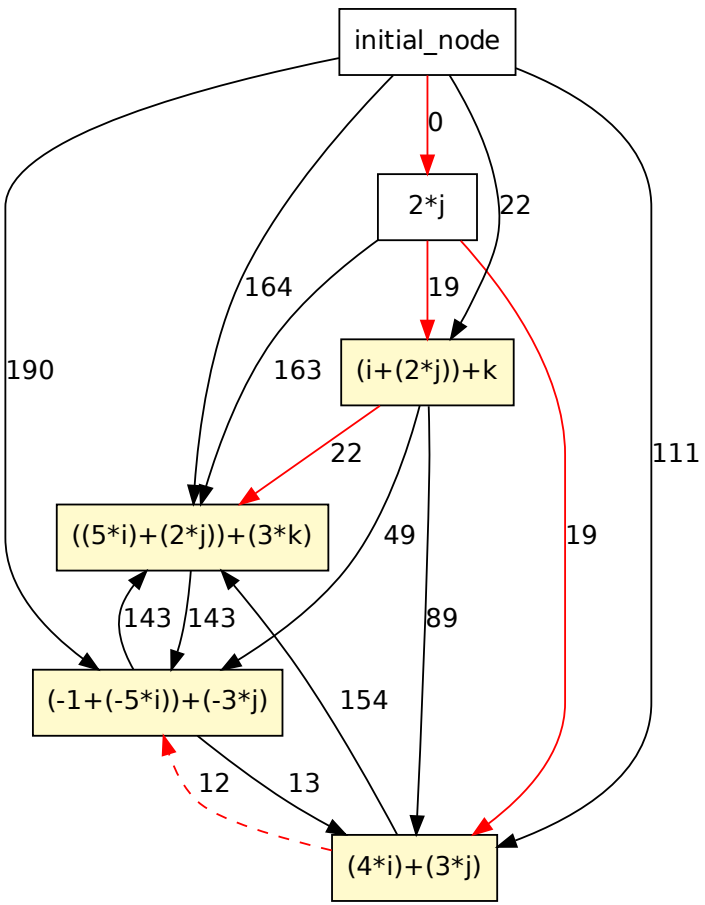
Id	Constraint
1	$i + 2j + k < 0$
2	$5i + 2j + 3k < 0$
3	$4i + 3j < 0$
4	$-5i - 3j - 1 < 0$

Edges: $u \xrightarrow{\Delta} v$:
v realized from u at cost Δ

Realization of v:
$initial\_node \xrightarrow{\Delta_1} u_1 \ldots \xrightarrow{\Delta_n} u_n \xrightarrow{\Delta} v$

Global realization:
spanning tree

Best realization:
minimum spanning tree



Experimental setup

Kernels: PolyBench/C v3.2

Target: FPGA Arria 10 10AX115S2F4I1SG

Synthesis: Intel Quartus Prime TM 16.1.2

Methodology:

- Control synthesis per process
- Compare:
  - SEM+Quartus: Our DAG, pipelined
  - Quartus: Direct implementation (affine control in VHDL) + output registers

Experimental results

Kernel	#dags	$\#\mathcal{C}$	$\#\mathcal{E}$	SEM+Quartus ALM	SEM+Quartus Regs	Quartus ALM	Quartus Regs	**Gain**
2mm	15	250	161	1011	612	1430	596	29%
3mm	18	369	206	1775	946	2528	922	30%
atax	12	134	83	628	328	900	321	30%
bicg	11	112	69	500	278	715	278	30%
correlation	27	356	205	1609	1129	2567	909	37%
covariance	16	243	143	1221	708	1872	618	35%
doitgen	9	145	124	393	280	607	268	35%
fdtd-2d	13	502	167	2339	1713	3293	1603	29%
gemm	10	125	93	672	270	851	270	21%
gemver	20	187	137	847	459	1102	438	23%
gesummv	14	95	84	456	245	549	227	**17%**
heat-3d	8	734	175	3545	1194	5667	2559	37%
jacobi-1d	8	134	64	628	556	912	520	31%
jacobi-2d	8	370	111	1660	1204	2547	1144	35%
lu	7	213	87	1116	666	1469	628	24%
mvt	11	118	70	550	290	758	290	27%
seidel-2d	5	226	63	1161	1464	1758	1291	34%
symm	13	213	116	1011	471	1540	465	34%
syr2k	10	135	90	721	290	944	281	24%
syrk	9	118	81	636	246	828	246	23%
trisolv	9	93	56	474	218	632	213	25%
trmm	8	118	79	549	262	806	253	32%

Maximum FPGA frequency (645 MHz), High compression ratio (5-6 input LUT)

`gesummv`: small bitwidth $\rightarrow$ low-level boolean optimizations are more effective

Outline

Conclusion

Models and algorithms for polyhedral HLS:

- DPN, a dataflow intermediate representation cross fertilizing dataflow models and partitioning
- Benefits: explicit data spilling, natural tuning of arithmetic intensity and parallelism
- Front-end and back-end compiler algorithms from C programs

$\Rightarrow$ Software: `dcc`, transferred to XtremLogic

Major concepts:

- (affine) tiling, **the** key transformation
- dataflow models, **the** key representation

Major locks:

- space/time scalability
- fork/join nature of polyhedral scheduling

Perspectives

Partial compilation

- Let parameters (parallelism, local footprint) survive the compilation
- Application: HLS/FPGA: tune the parallelism of a circuit
- Challenges: How to parametrize a DPN? What would be a generic parallel process?

Perspectives

Partial compilation

- Let parameters (parallelism, local footprint) survive the compilation
- Application: HLS/FPGA: tune the parallelism of a circuit
- Challenges: How to parametrize a DPN? What would be a generic parallel process?

Lazy compilation

- Complexity: set subtraction, min/max of piecewise affine mappings
- Idea: hide complexity with lazy values, evaluated dynamically (e.g. $R := P \setminus Q$)
- Challenges: How to compose/simplify lazy values? How to rephrase compiler analysis with lazy values?

Thanks to

Co-authors:

Fabrice Baray, Denis Barthou, Uday Bondhugula, Yongjian Chen, Alain Darte, Paul Feautrier, Carsten Fuhs, Laure Gonnord, Guobin He, Thomas Henretty, Guillaume Iooss, Sriram Krishnamoorthy, Haibo Lin, Qingda Lu, Tin-fook Ngai, Bogdan Pasca, Alexandru Plesco, Sanjay Rajopadhye, J. Ramanujam, Fabrice Rastello, Lawrence Rauchwerger, Atanas Rountev, Silvius Rus, P. Sadayappan, Yun Zou

PhD students:

Guillaume Iooss, Alexandru Plesco